

anki_vector.robot

This contains the `Robot` and `AsyncRobot` classes for managing Vector.

`Robot` will run all behaviors in sequence and directly return the results.

`AsyncRobot` will instead provide a `concurrent.futures.Future` which the caller may use to obtain the result when they desire.

Classes

<code>AsyncRobot</code> ([serial, ip, name, config, ...])	The AsyncRobot object is just like the Robot obj
<code>Robot</code> ([serial, ip, name, config, ...])	The Robot object is responsible for managing th

```
class anki_vector.robot.Robot(serial=None, ip=None, name=None, config=None,
default_logging=True, behavior_activation_timeout=10, cache_animation_lists=True,
enable_face_detection=False, estimate_facial_expression=False, enable_audio_feed=False,
enable_custom_object_detection=False, enable_nav_map_feed=None, show_viewer=False,
show_3d_viewer=False, behavior_control_level=<ControlPriorityLevel.DEFAULT_PRIORITY: 20>)
```

The Robot object is responsible for managing the state and connections to a Vector, and is typically the entry-point to running the sdk.

The majority of the robot will not work until it is properly connected to Vector. There are two ways to get connected:

1. Using `with`: it works just like opening a file, and will close when the `with` block's indentation ends.

```
import anki_vector

# Create the robot connection
with anki_vector.Robot() as robot:
    # Run your commands
    robot.anim.play_animation_trigger("GreetAfterLongTime")
```

2. Using `connect()` and `disconnect()` to explicitly open and close the connection: it allows the robot's connection to continue in the context in which it started.

```
import anki_vector

# Create a Robot object
robot = anki_vector.Robot()
# Connect to the Robot
robot.connect()
# Run your commands
robot.anim.play_animation_trigger("GreetAfterLongTime")
# Disconnect from Vector
robot.disconnect()
```

Parameters

- **serial** (`optional` [`str`]) – Vector’s serial number. The robot’s serial number (ex. 00e20100) is located on the underside of Vector, or accessible from Vector’s debug screen. Used to identify which Vector configuration to load.
- **ip** (`optional` [`str`]) – Vector’s IP address. (optional)
- **name** (`optional` [`str`]) – Vector’s name (in format `"Vector-XXXX"`) to be used for mDNS discovery. If a Vector with the given name is discovered, the `ip` parameter (and config field) will be overridden.
- **config** (`optional` [`dict`]) – A custom `dict` to override values in Vector’s configuration. (optional) Example:
`{"cert": "/path/to/file.cert", "name": "Vector-XXXX", "guid": "<secret_key>"}` where `cert` is the certificate to identify Vector, `name` is the name on Vector’s face when his backpack is double-clicked on the charger, and `guid` is the authorization token that identifies the SDK user. Note: Never share your authentication credentials with anyone.
- **default_logging** (`bool`) – Toggle default logging.
- **behavior_activation_timeout** (`int`) – The time to wait for control of the robot before failing.
- **cache_animation_lists** (`bool`) – Get the list of animation triggers and animations available at startup.
- **enable_face_detection** (`bool`) – Turn on face detection.
- **estimate_facial_expression** (`bool`) – Turn estimating facial expression on/off. Enabling `estimate_facial_expression` returns a facial expression, the expression values and the `anki_vector.util.ImageRect` for observed face regions (eyes, nose, and mouth) as part of the `RobotObservedFace` event. It is turned off by default as the number of `RobotObservedFace` events are reduced due to the increased processing time.
- **enable_audio_feed** (`bool`) – Turn audio feed on/off.
- **enable_custom_object_detection** (`bool`) – Turn custom object detection on/off.

- `enable_nav_map_feed` (`Optional` [`bool`]) – Turn navigation map feed on/off.
- `show_viewer` (`bool`) – Specifies whether to display a view of Vector’s camera in a window.
- `show_3d_viewer` (`bool`) – Specifies whether to display a 3D view of Vector’s understanding of the world in a window.
- `behavior_control_level` (`ControlPriorityLevel`) – Request control of Vector’s behavior system at a specific level of control. Pass `None` if behavior control is not needed. See `ControlPriorityLevel` for more information.

accel

The current accelerometer reading (x, y, z)

```
import anki_vector
with anki_vector.Robot() as robot:
    current_accel = robot.accel
```

Return type

`Vector3`

Type

`anki_vector.util.Vector3`

anim

A reference to the `AnimationComponent` instance.

Return type

`AnimationComponent`

audio

The audio instance used to control Vector’s microphone feed and speaker playback.

Return type

`AudioComponent`

behavior

A reference to the `BehaviorComponent` instance.

Return type

`BehaviorComponent`

camera

The `CameraComponent` instance used to control Vector's camera feed.

```
import anki_vector

with anki_vector.Robot() as robot:
    robot.camera.init_camera_feed()
    image = robot.camera.latest_image
    image.raw_image.show()
```

Return type

`CameraComponent`

carrying_object_id

The ID of the object currently being carried (-1 if none)

```
import anki_vector
from anki_vector.util import degrees

# Set the robot so that he can see a cube.
with anki_vector.Robot() as robot:
    robot.behavior.set_head_angle(degrees(0.0))
    robot.behavior.set_lift_height(0.0)

    robot.world.connect_cube()

    if robot.world.connected_light_cube:
        robot.behavior.pickup_object(robot.world.connected_light_cube)

    print("carrying_object_id: ", robot.carrying_object_id)
```

Return type

`int`

conn

A reference to the `Connection` instance.

Return type

`Connection`

connect(timeout=10)

Start the connection to Vector.

```
import anki_vector

robot = anki_vector.Robot()
robot.connect()
robot.anim.play_animation_trigger("GreetAfterLongTime")
robot.disconnect()
```

Parameters

timeout (`int`) – The time to allow for a connection before a `anki_vector.exceptions.VectorTimeoutException` is raised.

Return type

`None`

disconnect()

Close the connection with Vector.

```
import anki_vector

robot = anki_vector.Robot()
robot.connect()
robot.anim.play_animation_trigger("GreetAfterLongTime")
robot.disconnect()
```

Return type

`None`

enable_audio_feed

The audio feed enabled/disabled

Getter

Returns whether the audio feed is enabled

Setter

Enable/disable the audio feed

```
import asyncio
import time

import anki_vector

with anki_vector.Robot(enable_audio_feed=True) as robot:
    time.sleep(5)
    robot.enable_audio_feed = False
    time.sleep(5)
```

Return type

bool

events

A reference to the `EventHandler` instance.

Return type

EventHandler

faces

A reference to the `FaceComponent` instance.

Return type

FaceComponent

force_async

A flag used to determine if this is a `Robot` or `AsyncRobot`.

Return type

bool

get_battery_state()

Check the current state of the robot and cube batteries.

The robot is considered fully-charged above 4.1 volts. At 3.6V, the robot is approaching low charge.

Robot battery level values are as follows:

Value	Level	Description
1	Low	3.6V or less. If on charger, 4V or less.
2	Nominal	Normal operating levels.
3	Full	This state can only be achieved when Vector is on the charge

Cube battery level values are shown below:

Value	Level	Description
-------	-------	-------------

Value	Level	Description
1	Low	1.1V or less.
2	Normal	Normal operating levels.

```
import anki_vector

with anki_vector.Robot() as robot:
    print("Connecting to a cube...")
    robot.world.connect_cube()

    battery_state = robot.get_battery_state()
    if battery_state:
        print("Robot battery voltage: {}".format(battery_state.battery_volts))
        print("Robot battery Level: {}".format(battery_state.battery_level))
        print("Robot battery is charging: {}".format(battery_state.is_charging))
        print("Robot is on charger platform:
        {}".format(battery_state.is_on_charger_platform))
        print("Robot suggested charger time:
        {}".format(battery_state.suggested_charger_sec))
        print("Cube battery level: {}".format(battery_state.cube_battery.level))
        print("Cube battery voltage:
        {}".format(battery_state.cube_battery.battery_volts))
        print("Cube battery seconds since last reading:
        {}".format(battery_state.cube_battery.time_since_last_reading_sec))
        print("Cube battery factory id:
        {}".format(battery_state.cube_battery.factory_id))
```

Return type

BatteryStateResponse

get_version_state()

Get the versioning information for Vector, including Vector's os_version and engine_build_id.

```
import anki_vector

with anki_vector.Robot() as robot:
    version_state = robot.get_version_state()
    if version_state:
        print("Robot os_version: {}".format(version_state.os_version))
        print("Robot engine_build_id: {}".format(version_state.engine_build_id))
```

Return type

VersionStateResponse

gyro

The current gyroscope reading (x, y, z)

```
import anki_vector
with anki_vector.Robot() as robot:
    current_gyro = robot.gyro
```

Return type

Vector3

head_angle_rad

Vector's head angle (up/down).

```
import anki_vector
with anki_vector.Robot() as robot:
    current_head_angle_rad = robot.head_angle_rad
```

Return type

float

head_tracking_object_id

The ID of the object the head is tracking to (-1 if none)

```
import anki_vector
with anki_vector.Robot() as robot:
    current_head_tracking_object_id = robot.head_tracking_object_id
```

Return type

int

last_image_time_stamp

The robot's timestamp for the last image seen.

```
import anki_vector
with anki_vector.Robot() as robot:
    current_last_image_time_stamp = robot.last_image_time_stamp
```

Return type

int

left_wheel_speed_mmps

Vector's left wheel speed in mm/sec

```
import anki_vector
with anki_vector.Robot() as robot:
    current_left_wheel_speed_mmps = robot.left_wheel_speed_mmps
```

Return type

float

lift_height_mm

Height of Vector's lift from the ground.

```
import anki_vector
with anki_vector.Robot() as robot:
    current_lift_height_mm = robot.lift_height_mm
```

Return type

float

localized_to_object_id

The ID of the object that the robot is localized to (-1 if none)

```
import anki_vector
with anki_vector.Robot() as robot:
    current_localized_to_object_id = robot.localized_to_object_id
```

Return type

int

motors

A reference to the `MotorComponent` instance.

Return type

MotorComponent

nav_map

A reference to the `NavMapComponent` instance.

Return type

NavMapComponent

photos

A reference to the `PhotographComponent` instance.

Return type

`PhotographComponent`

pose

The current pose (position and orientation) of Vector.

```
import anki_vector
with anki_vector.Robot() as robot:
    current_robot_pose = robot.pose
```

Return type

`Pose`

Type

`anki_vector.util.Pose`

pose_angle_rad

Vector's pose angle (heading in X-Y plane).

```
import anki_vector
with anki_vector.Robot() as robot:
    current_pose_angle_rad = robot.pose_angle_rad
```

Return type

`float`

pose_pitch_rad

Vector's pose pitch (angle up/down).

```
import anki_vector
with anki_vector.Robot() as robot:
    current_pose_pitch_rad = robot.pose_pitch_rad
```

Return type

float

proximity

`ProximityComponent` containing state related to object proximity detection.

```
import anki_vector
with anki_vector.Robot() as robot:
    proximity_data = robot.proximity.last_sensor_reading
    if proximity_data is not None:
        print(proximity_data.distance)
```

Return type

`ProximityComponent`

right_wheel_speed_mmps

Vector's right wheel speed in mm/sec

```
import anki_vector
with anki_vector.Robot() as robot:
    current_right_wheel_speed_mmps = robot.right_wheel_speed_mmps
```

Return type

float

screen

A reference to the `ScreenComponent` instance.

Return type

`ScreenComponent`

status

A property that exposes various status properties of the robot.

This status provides a simple mechanism to, for example, detect if any of Vector's motors are moving, determine if Vector is being held, or if he is on the charger. The full list is available in the `RobotStatus` class documentation.

```
import anki_vector
with anki_vector.Robot() as robot:
    if robot.status.is_being_held:
        print("Vector is being held!")
    else:
        print("Vector is not being held.")
```

Return type

RobotStatus

touch

TouchComponent containing state related to object touch detection.

```
import anki_vector
with anki_vector.Robot() as robot:
    print('Robot is being touched:
{0}'.format(robot.touch.last_sensor_reading.is_being_touched))
```

Return type

TouchComponent

viewer

The **ViewerComponent** instance used to render Vector's camera feed.

```
import time

import anki_vector

with anki_vector.Robot() as robot:
    # Render video for 5 seconds
    robot.viewer.show()
    time.sleep(5)

    # Disable video render and camera feed for 5 seconds
    robot.viewer.close()
```

Return type

ViewerComponent

viewer_3d

The **Viewer3DComponent** instance used to render Vector's navigation map.

```
import time

import anki_vector

with anki_vector.Robot(show_3d_viewer=True, enable_nav_map_feed=True) as robot:
    # Render 3D view of navigation map for 5 seconds
    time.sleep(5)
```

Return type

Viewer3DComponent

vision

VisionComponent containing functionality related to vision based object detection.

```
import anki_vector
with anki_vector.Robot() as robot:
    robot.vision.enable_custom_object_detection()
```

Return type

VisionComponent

world

A reference to the World instance, or None if the World is not yet initialized.

Return type

World

`class anki_vector.robot.AsyncRobot(serial=None, ip=None, name=None, config=None, default_logging=True, behavior_activation_timeout=10, cache_animation_lists=True, enable_face_detection=False, estimate_facial_expression=False, enable_audio_feed=False, enable_custom_object_detection=False, enable_nav_map_feed=None, show_viewer=False, show_3d_viewer=False, behavior_control_level=<ControlPriorityLevel.DEFAULT_PRIORITY: 20>)`

The AsyncRobot object is just like the Robot object, but allows multiple commands to be executed at the same time. To achieve this, all grpc function calls also return a

`concurrent.futures.Future`.

1. Using `with`: it works just like opening a file, and will close when the `with` block's indentation ends.

```
import anki_vector
from anki_vector.util import degrees

# Create the robot connection
with anki_vector.AsyncRobot() as robot:
    # Start saying text asynchronously
    say_future = robot.behavior.say_text("Now is the time")
    # Turn robot, wait for completion
    turn_future = robot.behavior.turn_in_place(degrees(3*360))
    turn_future.result()
    # Play greet animation trigger, wait for completion
    greet_future = robot.anim.play_animation_trigger("GreetAfterLongTime")
    greet_future.result()
    # Make sure text has been spoken
    say_future.result()
```

2. Using `connect()` and `disconnect()` to explicitly open and close the connection: it allows the robot's connection to continue in the context in which it started.

```
import anki_vector
from anki_vector.util import degrees

# Create a Robot object
robot = anki_vector.AsyncRobot()
# Connect to Vector
robot.connect()
# Start saying text asynchronously
say_future = robot.behavior.say_text("Now is the time")
# Turn robot, wait for completion
turn_future = robot.behavior.turn_in_place(degrees(3 * 360))
turn_future.result()
# Play greet animation trigger, wait for completion
greet_future = robot.anim.play_animation_trigger("GreetAfterLongTime")
greet_future.result()
# Make sure text has been spoken
say_future.result()
# Disconnect from Vector
robot.disconnect()
```

When getting callbacks from the event stream, it's important to understand that function calls return a `concurrent.futures.Future` and not an `asyncio.Future`. This means any async callback functions will need to use `asyncio.wrap_future()` to be able to await the function's response.

```
import asyncio
import time

import anki_vector

async def callback(robot, event_type, event):
    await asyncio.wrap_future(robot.anim.play_animation_trigger('GreetAfterLongTime'))
    await asyncio.wrap_future(robot.behavior.set_head_angle(anki_vector.util.degrees(40)))

if __name__ == "__main__":
    args = anki_vector.util.parse_command_args()
    with anki_vector.AsyncRobot(serial=args.serial, enable_face_detection=True) as robot:
        robot.behavior.set_head_angle(anki_vector.util.degrees(40))
        robot.events.subscribe(callback, anki_vector.events.Events.robot_observed_face)

    # Waits 10 seconds. Show Vector your face.
    time.sleep(10)
```

Parameters

- **serial** (`Optional` [`str`]) – Vector's serial number. The robot's serial number (ex. 00e20100) is located on the underside of Vector, or accessible from Vector's debug screen. Used to identify which Vector configuration to load.
- **ip** (`Optional` [`str`]) – Vector's IP Address. (optional)

- **config** (`Optional [ dict ]`) – A custom `dict` to override values in Vector's configuration. (optional) Example:

```
{"cert": "/path/to/file.cert", "name": "Vector-XXXX", "guid": "<secret_key>"}
```

where `cert` is the certificate to identify Vector, `name` is the name on Vector's face when his backpack is double-clicked on the charger, and `guid` is the authorization token that identifies the SDK user. Note: Never share your authentication credentials with anyone.
- **default_logging** (`bool`) – Toggle default logging.
- **behavior_activation_timeout** (`int`) – The time to wait for control of the robot before failing.
- **cache_animation_lists** (`bool`) – Get the list of animation triggers and animations available at startup.
- **enable_face_detection** (`bool`) – Turn on face detection.
- **estimate_facial_expression** (`bool`) – Turn estimating facial expression on/off.
- **enable_audio_feed** (`bool`) – Turn audio feed on/off.
- **enable_custom_object_detection** (`bool`) – Turn custom object detection on/off.
- **enable_nav_map_feed** (`Optional [ bool ]`) – Turn navigation map feed on/off.
- **show_viewer** (`bool`) – Specifies whether to display a view of Vector's camera in a window.
- **show_3d_viewer** (`bool`) – Specifies whether to display a 3D view of Vector's understanding of the world in a window.
- **behavior_control_level** (`ControlPriorityLevel`) – Request control of Vector's behavior system at a specific level of control. Pass `None` if behavior control is not needed. See `ControlPriorityLevel` for more information.